Title: Covid - 19 Resource Delivery System

2021

Submitted By:

1. Ananya Chopra (20103127)
2. Drishtant Maurya (20103140)
3. Nandini Agarwal (20103144)
4. Vivek Arora (20103145)

Submitted To:

Mr. Satish Chandra

Contents

Introduction

Basically, the code would be programmed in such a way that it is able to deliver us the required

information. The availability of resources which are required for the treatment of covid-19 and, is logistically the best and the most efficient route for the Covid-19 Resources to get delivered to the hospitals which are in need, from the nearest Warehouses.

In this project, we have considered the map of Noida. We have taken 8 busiest hospitals, where the number of patients administering on a daily basis is above 100, and 3 warehouses which are handling the maximum workload of the area and where the Resources are stored.

Initially, the user would be asked for the Hospital's name, where the resources are required. Then, the program would ask for what resources are required in that hospital, for example, oxygen cylinders, injections, medicines, etc. The owner of the warehouse will input the amount of the respective resources their warehouse has in stock.

This is where the main objective of the code comes into play. Now, the program would display the availability of different items which are in stock in different warehouses, and the best possible route from the Warehouse to the Hospital. The availability of the specific item and the Total Time the resource would take to get delivered to their destined location would always be calculated and displayed.

The time shown would also consider all the traffic on these routes. And, The Traffic's Congestion Level would be divided in 10 Levels, with Level 10 Traffic being the most congested one. We would be using linked list, graphs, OOPS to execute the tasks mentioned above.

Data Structures

1) **Heaps:** It will be used in Dijkstra's Algorithm

2) **Linked List:** It will be used in storing nodes of linked list.

3) Graphs: Used for making maps

4) Adjacency Matrix or Adjacency List: It will be used for graph storage

5) Other Data Structures might be added as per the requirement while building the project code.

6) OOPS: used for storing information of various assets involved in the process.

Algorithms Used

1. Dijkstra's Algorithm.

a. Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph.

b. It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.

2. Sorting Algorithms.

a. A sorting algorithm is a method for reorganizing a large number of items into a specific order, such as alphabetical, highest-to-lowest value or shortest-to-longest distance.

b. Sorting algorithms take lists of items as input data, perform specific operations on those lists and deliver ordered arrays as output.

3. Searching Algorithms.

a. A search algorithm is the step-by-step procedure used to locate specific data among a collection of data.

b. It is considered a fundamental procedure in computing. In computer science, when searching for data, the difference between a fast application and a slower one often lies in the use of the proper search algorithm.

Software Used

1. Visual Studio Code:

Visual Studio Code is a free source-code editor made by Microsoft for Windows, Linux and macOS.

2. GitBash:

Git Bash is a source control management system for Windows. It allows users to type Git commands that make source code management easier through versioning and commit history.

3. MinGW:

It formerly mingw32,is free and open source software development environment to create Microsoft Windows application

4. Github:

It makes it easy to contribute to your group work projects,
It helps in documentation, It tracks changes in your code across versions

Implementation:

Code:

```
#include <iostream>

#include <ctime>

//Total assets including both hospitals and warehouse

#define Assets 11

using namespace std;

struct Information

{

    int Id;

    string Name, Address;

    string Type;
```

```
} Info[Assets] = {{0, "SJM", "Sector 63", "h"},
                 {1, "Prakash", "Sector 33", "h"},
                 {2, "Jaypee", "Sector 128", "h"},
                 {3, "Max", "Sector 19", "h"},
                 {4, "Yatharth", "Sector 110", "h"},
                 {5, "NMC", "Sector 30", "h"},
                 {6, "Kailash", "Sector 27", "h"},
                 {7, "Apollo", "Sector 26", "h"},
                 {8, "Singh", " Sector 4", " W "},
                 {9, "Mathura", "Sector 62", " W "},
                 {10, "Maheshwari", "Sector 69", "w"}};


// Data structure to store a graph edge
struct Node
{
    int Val;
    Information *Data;
    Node *Next;
    int Traffic;
};
// Function to print all neighboring vertices of a given
vertex
void printList(Node *ptr)
{
    while (ptr != nullptr)
    {
```

```
        cout << " Road to ";
        if (ptr->Data->Type == "h")
            cout << "hospital -";
        else
            cout << "warehouse -";
        cout << ptr->Data->Name << " has traffic level " <<
ptr->Traffic << "\n";
        ptr = ptr->Next;
    }
    cout << endl;
}


// Data structure to store a graph edge
struct Edge
{
    int Source, Destination, Traffic;
};


class Graph
{
    // Function to allocate a new node for the adjacency list
    Node *GetAdjListNode(int Destination, Node *Head, int
Traffic)
    {
        Node *newNode = new Node;
        newNode->Val = Destination;
```

```
        // point new node to the current Head
        newNode->Next = Head;
        newNode->Traffic = Traffic;
        newNode->Data = &Info[Destination];
        return newNode;
    }

    int N; // total number of nodes in the graph

public:
    // An array of pointers to Node to represent the
    // adjacency list
    Node **Head;

    // Constructor
    Graph(Edge Edges[], int n /*number of edges*/, int N)
    {
        // allocate memory
        Head = new Node *[N]();
        this->N = N;

        // initialize Head pointer for all vertices
        for (int i = 0; i < N; i++)
        {
```

```
            Head[i] = nullptr;
        }

        // add Edges to the directed graph
        for (unsigned i = 0; i < n; i++)
        {
            int Source = Edges[i].Source;
            int Destination = Edges[i].Destination;
            int Traffic = Edges[i].Traffic;
            // insert at the beginning
            Node *newNode = GetAdjListNode(Destination,
Head[Source], Traffic);

            // point Head pointer to the new node
            Head[Source] = newNode;

            // uncomment the following code for undirected
graph

            newNode = GetAdjListNode(Source,
Head[Destination], Traffic);

            // change Head pointer to point to the new node
            Head[Destination] = newNode;
        }
    }
```

```
    void printGraph()
    {
        // print adjacency list representation of a graph
        for (int i = 0; i < N; i++)
        {
            // print given vertex
            cout << "Starting from " << Info[i].Name << ":
\n";

            // print all its neighboring vertices
            printList(this->Head[i]);
        }
    }

    int findDestinationId(string name, int a, int b, int c)
    {
        for (int i = 0; i < Assets; i++)
        {
            if (Info[i].Name == name)
            {
                return Info[i].Id;
            }
        }
        return -1;
    }
```

```
    int Vertices()
    {
        return N;
    }
    // Destinationructor
    ~Graph()
    {
        for (int i = 0; i < N; i++)
        {
            delete[] Head[i];
        }

        delete[] Head;
    }
};

int random(int min, int max)
{
    int random_variable = rand();
    return min + (random_variable % (max - min + 1));
}

// Structure to represent a min heap node
struct MinHeapNode
```

```
{
    int v;
    int dist;
};

// Structure to represent a min heap
struct MinHeap
{

    // Number of heap nodes present currently
    int size;

    // Capacity of min heap
    int capacity;

    // This is needed for decreaseKey()
    int *pos;
    struct MinHeapNode **array;
};

// A utility function to create a
// new Min Heap Node
struct MinHeapNode *newMinHeapNode(int v,
                                   int dist)
{
```

```
    struct MinHeapNode *minHeapNode =
        (struct MinHeapNode *)
            malloc(sizeof(struct MinHeapNode));
    minHeapNode->v = v;
    minHeapNode->dist = dist;
    return minHeapNode;
}


// A utility function to create a Min Heap
struct MinHeap *createMinHeap(int capacity)
{
    struct MinHeap *minHeap =
        (struct MinHeap *)
            malloc(sizeof(struct MinHeap));
    minHeap->pos = (int *)malloc(
        capacity * sizeof(int));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array =
        (struct MinHeapNode **)
            malloc(capacity *
                   sizeof(struct MinHeapNode *));
    return minHeap;
}
```

```
// A utility function to swap two
// nodes of min heap.
// Needed for min heapify
void swapMinHeapNode(struct MinHeapNode **a,
                     struct MinHeapNode **b)
{
    struct MinHeapNode *t = *a;
    *a = *b;
    *b = t;
}


// A standard function to
// heapify at given idx
// This function also updates
// position of nodes when they are swapped.
// Position is needed for decreaseKey()
void minHeapify(struct MinHeap *minHeap,
                int idx)
{
    int smallest, left, right;
    smallest = idx;
    left = 2 * idx + 1;
    right = 2 * idx + 2;


    if (left < minHeap->size &&
```

```
        minHeap->array[left]->dist <
            minHeap->array[smallest]->dist)
        smallest = left;

    if (right < minHeap->size &&
        minHeap->array[right]->dist <
            minHeap->array[smallest]->dist)
        smallest = right;

    if (smallest != idx)
    {
        // The nodes to be swapped in min heap
        MinHeapNode *smallestNode =
            minHeap->array[smallest];
        MinHeapNode *idxNode =
            minHeap->array[idx];

        // Swap positions
        minHeap->pos[smallestNode->v] = idx;
        minHeap->pos[idxNode->v] = smallest;

        // Swap nodes
        swapMinHeapNode(&minHeap->array[smallest],
                        &minHeap->array[idx]);
```

```
        minHeapify(minHeap, smallest);
    }
}

// A utility function to check if
// the given minHeap is ampty or not
int isEmpty(struct MinHeap *minHeap)
{
    return minHeap->size == 0;
}

// Standard function to extract
// minimum node from heap
struct MinHeapNode *extractMin(struct MinHeap *
                                    minHeap)
{
    if (isEmpty(minHeap))
        return NULL;

    // Store the root node
    struct MinHeapNode *root =
        minHeap->array[0];

    // Replace root node with last node
    struct MinHeapNode *lastNode =
```

```
        minHeap->array[minHeap->size - 1];
    minHeap->array[0] = lastNode;

    // Update position of last node
    minHeap->pos[root->v] = minHeap->size - 1;
    minHeap->pos[lastNode->v] = 0;

    // Reduce heap size and heapify root
    --minHeap->size;
    minHeapify(minHeap, 0);

    return root;
}

// Function to decreasy dist value
// of a given vertex v. This function
// uses pos[] of min heap to get the
// current index of node in min heap
void decreaseKey(struct MinHeap *minHeap,
                 int v, int dist)
{
    // Get the index of v in  heap array
    int i = minHeap->pos[v];

    // Get the node and update its dist value
```

```
    minHeap->array[i]->dist = dist;

    // Travel up while the complete
    // tree is not hepified.
    // This is a O(Logn) loop
    while (i && minHeap->array[i]->dist <
                    minHeap->array[(i - 1) / 2]->dist)
    {
        // Swap this node with its parent
        minHeap->pos[minHeap->array[i]->v] =
            (i - 1) / 2;
        minHeap->pos[minHeap->array[(i - 1) / 2]->v] = i;
        swapMinHeapNode(&minHeap->array[i],
                        &minHeap->array[(i - 1) / 2]);

        // move to parent index
        i = (i - 1) / 2;
    }
}

// A utility function to check if a given vertex
// 'v' is in min heap or not
bool isInMinHeap(struct MinHeap *minHeap, int v)
{
    if (minHeap->pos[v] < minHeap->size)
```

```
        return true;
    return false;
}
int convert(int val)
{
    return 5 * val;
}

struct Warehouse
{
    int Id;
    int dist;
};

// A utility function used to print the solution
void SmallestRoute(int dist[], int n, int src, int dest)
{
    cout << "Time taken from warehouse " << Info[src].Name <<
" to reach hospital " << Info[dest].Name << " is " <<
convert(dist[dest]) << " minutes." << endl;
}

// The main function that calculates
// distances of shortest paths from src to all
// vertices. It is a O(ELogV) function
Warehouse dijkstra(Graph *graph, int src, int dest)
```

```
{


    // Get the number of vertices in graph
    int V = graph->Vertices();


    // dist values used to pick
    // minimum weight edge in cuh
    int dist[V];


    // minHeap represents set E
    struct MinHeap *minHeap = createMinHeap(V);


    // Initialize min heap with all
    // vertices. dist value of all vertices
    for (int v = 0; v < V; ++v)
    {
        dist[v] = INT32_MAX;
        minHeap->array[v] = newMinHeapNode(v, dist[v]);
        minHeap->pos[v] = v;
    }


    // Make dist value of src vertex
    // as 0 so that it is extracted first
    minHeap->array[src] =
        newMinHeapNode(src, dist[src]);
```

```
    minHeap->pos[src] = src;
    dist[src] = 0;
    decreaseKey(minHeap, src, dist[src]);

    // Initially size of min heap is equal to V
    minHeap->size = V;

    // In the followin loop,
    // min heap contains all nodes
    // whose shortest distance
    // is not yet finalized.
    while (!isEmpty(minHeap))
    {
        // Extract the vertex with
        // minimum distance value
        struct MinHeapNode *minHeapNode =
            extractMin(minHeap);

        // Store the extracted vertex number
        int u = minHeapNode->v;

        // Traverse through all adjacent
        // vertices of u (the extracted
        // vertex) and update
        // their distance values
```

```
        Node *pCrawl =
            graph->Head[u];
        while (pCrawl != NULL)
        {
            int v = pCrawl->Val;

            // If shortest distance to v is
            // not finalized yet, and distance to v
            // through u is less than its
            // previously calculated distance
            if (isInMinHeap(minHeap, v) &&
                dist[u] != INT32_MAX &&
                pCrawl->Traffic + dist[u] < dist[v])
            {
                dist[v] = dist[u] + pCrawl->Traffic;

                // update distance
                // value in min heap also
                decreaseKey(minHeap, v, dist[v]);
            }
            pCrawl = pCrawl->Next;
        }
    }

    // print the calculated shortest distances
```

```
    SmallestRoute(dist, V, src, dest);
    Warehouse w = {src, dist[dest]};
    return w;
}
/*
struct Edge
{
    int Source, Destination, Traffic;
};
*/
Graph *CreateMap()
{
    Edge Edges[19] =
        {
            // pair `(x, y)` represents an edge from `x` to
`y`
            // 11 vertices because total of warehouse and
hospital
            {0, 5},
            {1, 5},
            {1, 2},
            {2, 3},
            {2, 4},
            {3, 5},
            {3, 6},
            {4, 7},
```

```
            {4, 8},
            {5, 9},
            {5, 10},
            {6, 5},
            {7, 6},
            {7, 10},
            {8, 7},
            {8, 0},
            {9, 2},
            {9, 3},
            {10, 9}
            };
    for (int i = 0; i < sizeof(Edges) / sizeof(Edges[0]);
i++)
    {
        Edges[i].Traffic = random(1, 10);
    }
    // total number of nodes in the graph
    // total number of assets.-->hospital and warehouse
    int N = 11;

    // calculate the total number of Edges
    int n = 19;

    // construct graph
    Graph *Graphic = new Graph(Edges, n, N);
```

```
    return Graphic;
}

//--Graph implementation in C++ without using STL
int main()
{
    srand(time(nullptr)); //use current time as seed for random generator

    Graph *G = CreateMap();

    cout << " -------- Welcome to Covid-19 Resource Delivery System ----------- \n\n";

    cout << "For the purpose of this demonstration, we have considered 8 hospitals and 3 warehouses\n";

    cout << "Details of the above is as follows : \n";

    for (int i = 0; i < Assets; i++)
    {
        if (Info[i].Type == "h")
        {
            cout << "Hospital : " << Info[i].Name << ", " << Info[i].Address << endl;
        }
```

```
        else
        {
            cout << "Warehouse : " << Info[i].Name << ", " <<
Info[i].Address << endl;
        }
    }
    cout << "\n";
    G->printGraph();

    cout << "Enter the name of hospital that require
resources : ";
    string hospital;
    cin >> hospital;

    int a, b, c;
    cout << "Enter the required no of oxygen
cylinders/concentrator : ";
    cin >> a;
    cout << "Enter the required no of PPE kit : ";
    cin >> b;
    cout << "Enter the required no of remdesivir : ";
    cin >> c;

    int dest = G->findDestinationId(hospital, a, b, c);
    cout << endl;
```

```
    if (dest == -1)
    {
        cout << "Invalid hospital name. Please enter a valid
hospital name." << endl;
        return 0;
    }

    Warehouse fastest = {-1, INT32_MAX};

    for (int i = 8; i < 11; i++)
    {
        Warehouse temp = dijkstra(G, Info[i].Id, dest);
        if (temp.dist < fastest.dist)
            fastest = temp;
    }

    cout << "\nAmong these, the most efficient warehouse for
delivering the required resources is : ";

    cout << Info[fastest.Id].Name << endl;

    return 0;
}
```

Output:-

-------- Welcome to Covid-19 Resource Delivery System -----------

For the purpose of this demonstration, we have considered 8 hospitals and 3 warehouses
Details of the above is as follows :
Hospital : SJM, Sector 63
Hospital : Prakash, Sector 33
Hospital : Jaypee, Sector 128
Hospital : Max, Sector 19
Hospital : Yatharth, Sector 110
Hospital : NMC, Sector 30
Hospital : Kailash, Sector 27
Hospital : Apollo, Sector 26
Warehouse : Singh, Sector 4
Warehouse : Mathura, Sector 62
Warehouse : Maheshwari, Sector 69

```
Starting from SJM:
 Road to warehouse -Singh has traffic level 4
 Road to hospital -NMC has traffic level 3

Starting from Prakash:
 Road to hospital -Jaypee has traffic level 3
 Road to hospital -NMC has traffic level 1

Starting from Jaypee:
 Road to warehouse -Mathura has traffic level 1
 Road to hospital -Yatharth has traffic level 8
 Road to hospital -Max has traffic level 7
 Road to hospital -Prakash has traffic level 3

Starting from Max:
 Road to warehouse -Mathura has traffic level 7
 Road to hospital -Kailash has traffic level 9
 Road to hospital -NMC has traffic level 6
 Road to hospital -Jaypee has traffic level 7

Starting from Yatharth:
 Road to warehouse -Singh has traffic level 1
 Road to hospital -Apollo has traffic level 6
 Road to hospital -Jaypee has traffic level 8

Starting from Kailash:
 Road to hospital -Apollo has traffic level 3
 Road to hospital -NMC has traffic level 1
 Road to hospital -Max has traffic level 9

Starting from Apollo:
 Road to warehouse -Singh has traffic level 1
 Road to warehouse -Maheshwari has traffic level 10
 Road to hospital -Kailash has traffic level 3
 Road to hospital -Yatharth has traffic level 6

Starting from Singh:
 Road to hospital -SJM has traffic level 4
 Road to hospital -Apollo has traffic level 1
 Road to hospital -Yatharth has traffic level 1

Starting from Mathura:
 Road to warehouse -Maheshwari has traffic level 6
 Road to hospital -Max has traffic level 7
 Road to hospital -Jaypee has traffic level 1
 Road to hospital -NMC has traffic level 8

Starting from Maheshwari:
 Road to warehouse -Mathura has traffic level 6
 Road to hospital -Apollo has traffic level 10
 Road to hospital -NMC has traffic level 4
```

```
Enter the name of hospital that require resources : Apollo
Enter the required no of oxygen cylinders/concentrator : 5
Enter the required no of PPE kit : 4
Enter the required no of remdesivir : 7

Time taken from warehouse Singh to reach hospital Apollo is 5 minutes.
Time taken from warehouse Mathura to reach hospital Apollo is 45 minutes.
Time taken from warehouse Maheshwari to reach hospital Apollo is 40 minutes.
```